AMD DEVELOPER symposium 2002

NAG High Performance Numerical Libraries

Mick Pont, NAG, Oxford

Numerical Algorithms Group

AMD DEVELOPER symposium 2002

- Founded 1970
  - Co-operative software project
- Incorporated 1976
  - NAG Ltd. (UK)
  - Not-for-profit organisation
- Offices:
  - NAG Ltd., Oxford, UK
  - NAG Inc., Chicago, USA
  - NAG GmbH, Nr. Munich, Germany
  - Nihon NAG KK, Tokyo, Japan

Key NAG Products

AMD DEVELOPER symposium 2002

- Numerical and Statistical Libraries
  - Over 1200 user-callable components
- Visualization
- Compiler and associated tools
- Data Mining Components
- Consultancy Services

Numerical Libraries

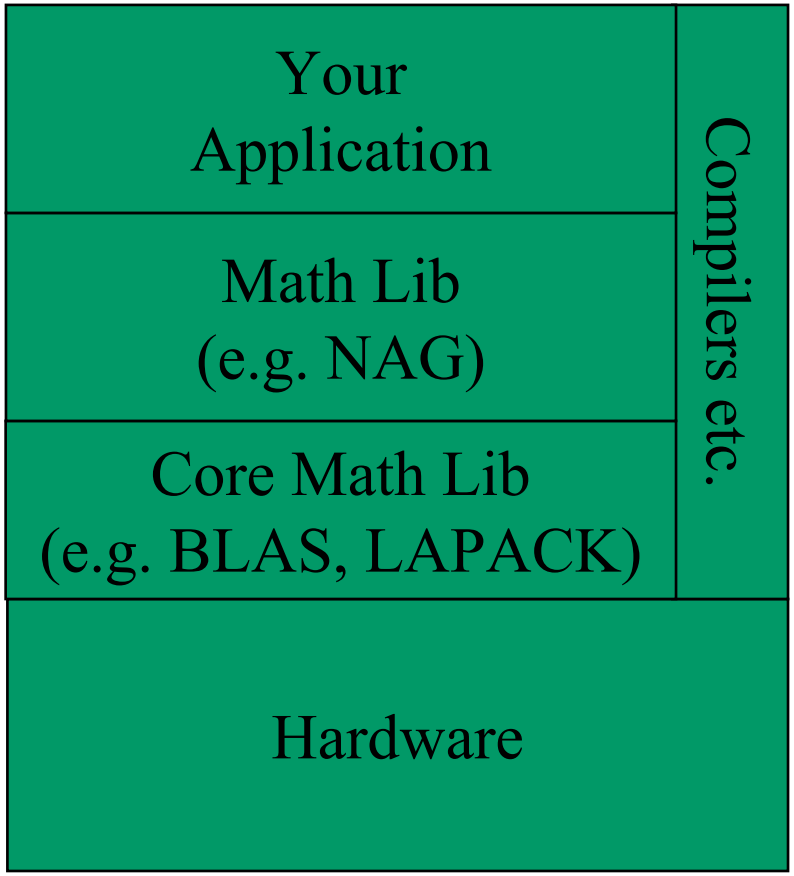
AMD DEVELOPER symposium 2002

- Nonlinear equation solvers
- Summation of series and transformations, FFTs
- Quadrature
- ODEs, PDEs and integral equations
- Approximation and curve and surface fitting
- Optimization and operations research
- Dense linear algebra, including LAPACK
- Sparse linear systems and eigenproblems
- Special functions

Achieving performance

AMD DEVELOPER symposium 2002

- Multiplicative effect of improvements
  - Hardware advances…
  - Better compilers
  - Optimize the core
  - Provide/optimize useful functionality
  - Take full advantage of future improvements at ANY level



NAG SMP Library

AMD DEVELOPER symposium 2002

- Includes the full functionality of over 1200 routines in the NAG Fortran Library
- Significant speed-ups over the NAG Fortran Library and Hardware Vendors' LAPACK implementations
  E.g. SVD vs IBM* ESSL* (using ESSL* BLAS)
  - 9x better on single processor
  - 35x better on 4 processors
- Proven scalability with over 64 processors

* Names and brands may be claimed as the property of others

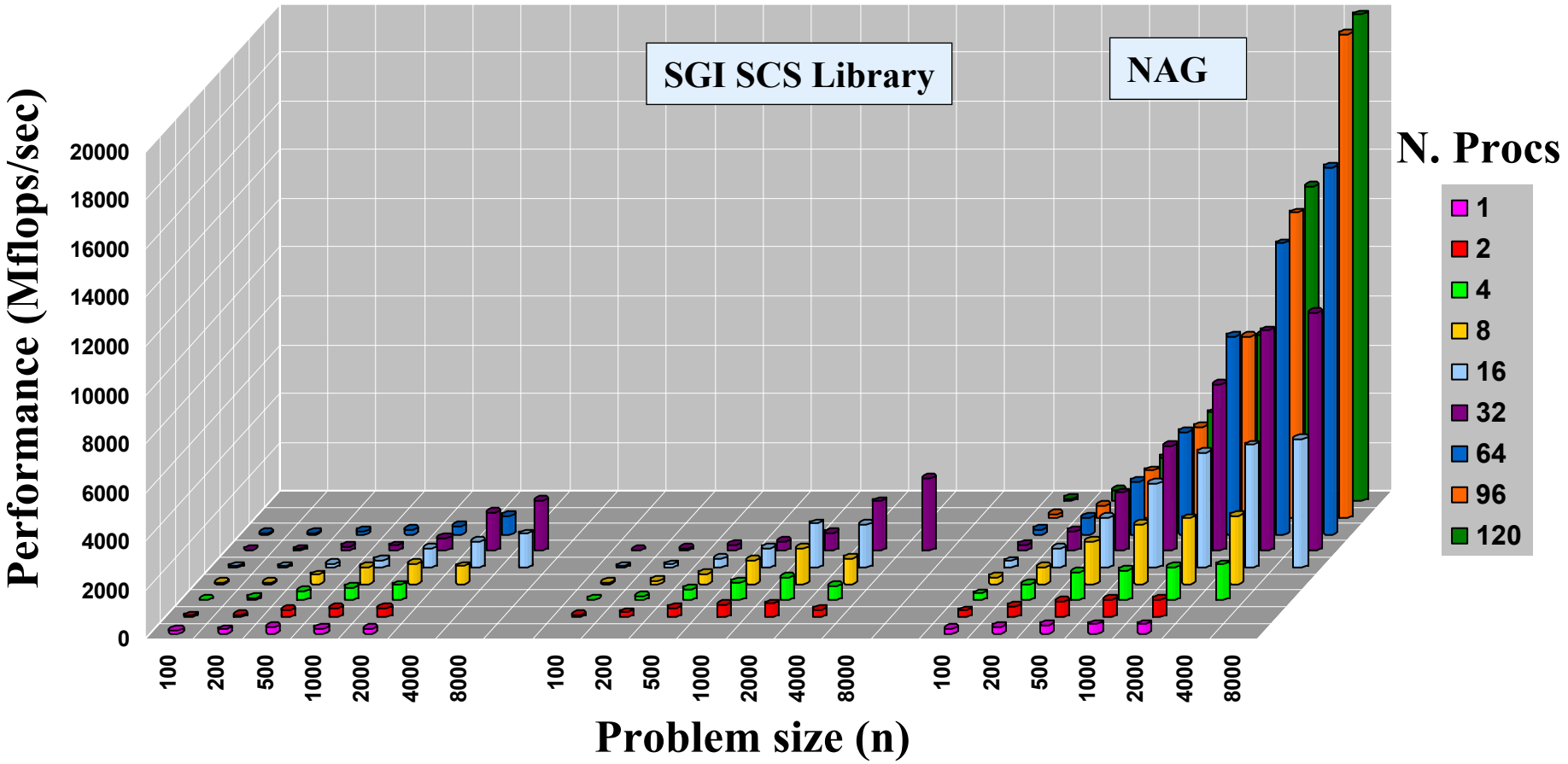
NAG SMP Library - Overview

AMD DEVELOPER symposium 2002

- Over 240 tuned and enhanced routines
- Same interfaces as FL, just link 'n' go
- Major areas benefitting:
  - Linear Algebra
  - Multivariate statistics
  - FFTs
  - Optimization
  - Sparse Matrix Computations
  - Random Number Generation

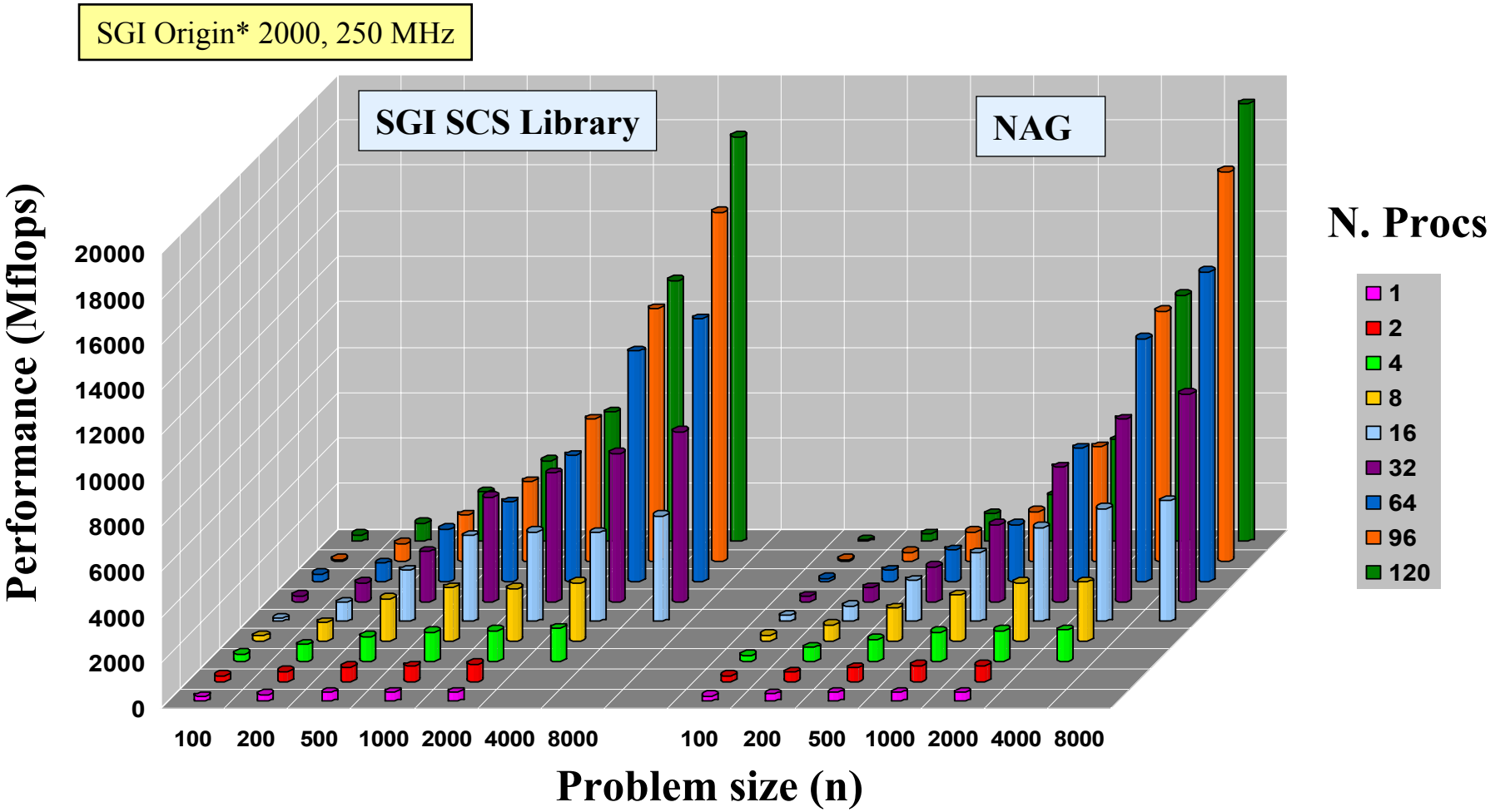
SGI* Origin*: QR Factorization

SGI Origin* 2000, 250 MHz



* Names and brands may be claimed as the property of others

LU Factorisation
(Linpack Benchmark effect)



* Names and brands may be claimed as the property of others

LU Factorization

AMD DEVELOPER symposium 2002

For a given matrix A, find a lower triangular matrix L and an upper triangular matrix U such that LU = A. e.g.

$$A = \begin{matrix} 3 & 5 \\ 6 & 7 \end{matrix} \qquad L = \begin{matrix} 1 & 0 \\ 2 & 1 \end{matrix} \qquad U = \begin{matrix} 3 & 5 \\ 0 & -3 \end{matrix}$$

Used to solve simultaneous linear equations:

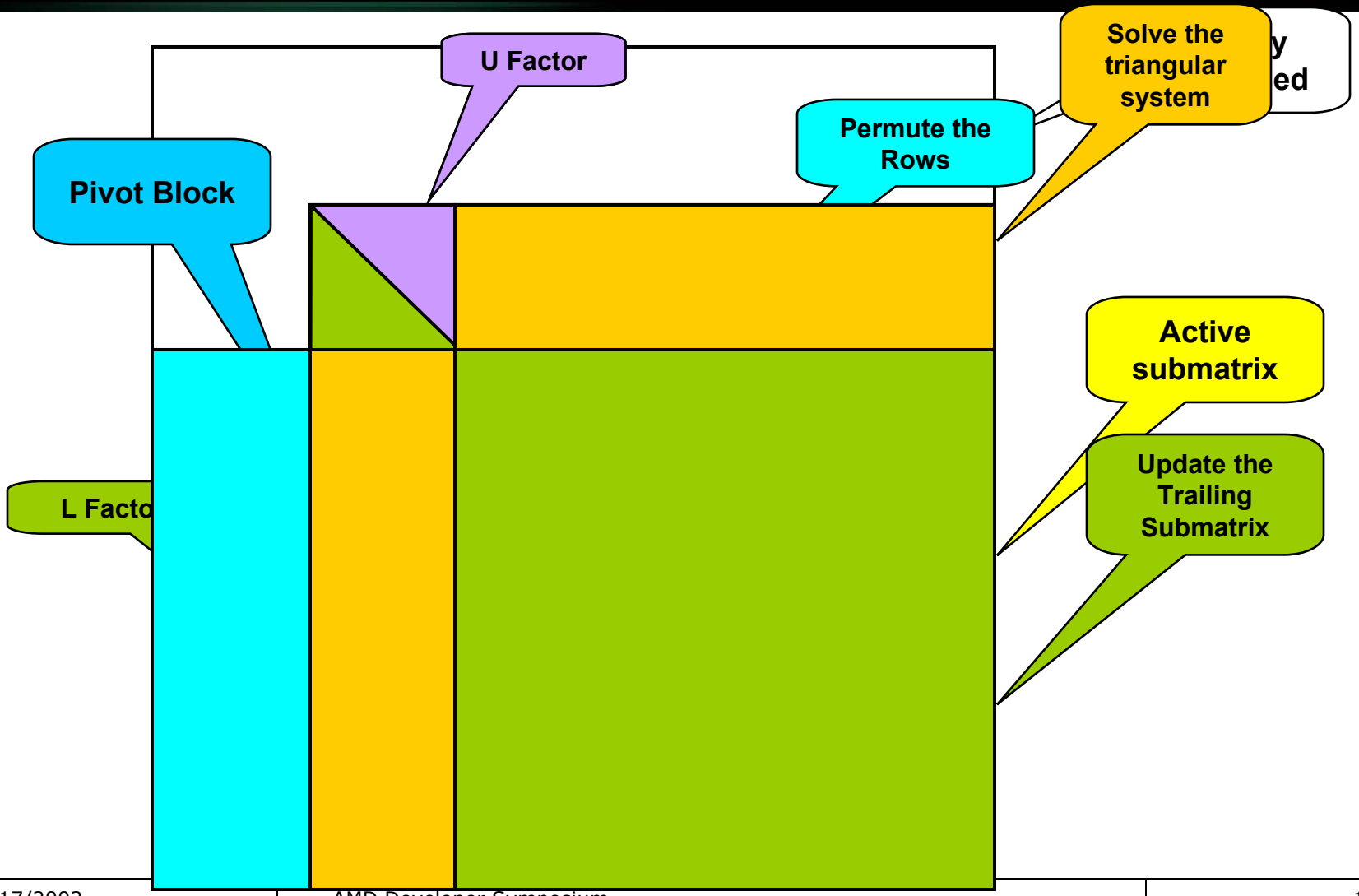
$$\begin{matrix} & 3x + 5y & = 9 \\ & 6x + 7y & = 4 \\ \\ \Rightarrow & 3x + 5y & = 9 \\ & -3y & = -14 \end{matrix}$$

Anatomy of the LU Factorization

AMD DEVELOPER symposium 2002

- Identify serial bottlenecks
- Identify memory access bottlenecks
- Remove the above or …
- "Hide" them using a "look ahead" strategy

LU Factorisation: LAPACK Style

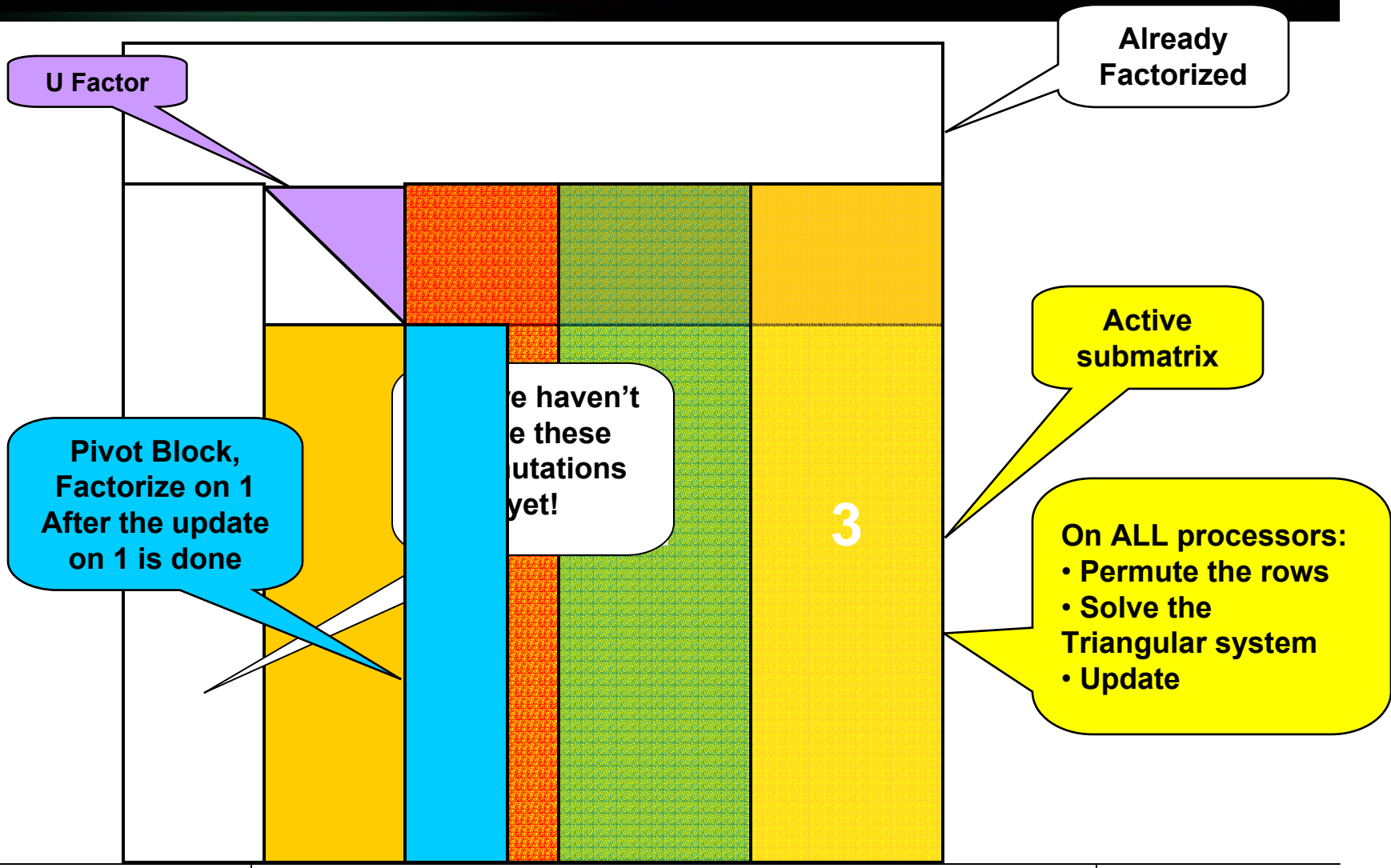


The slow bits

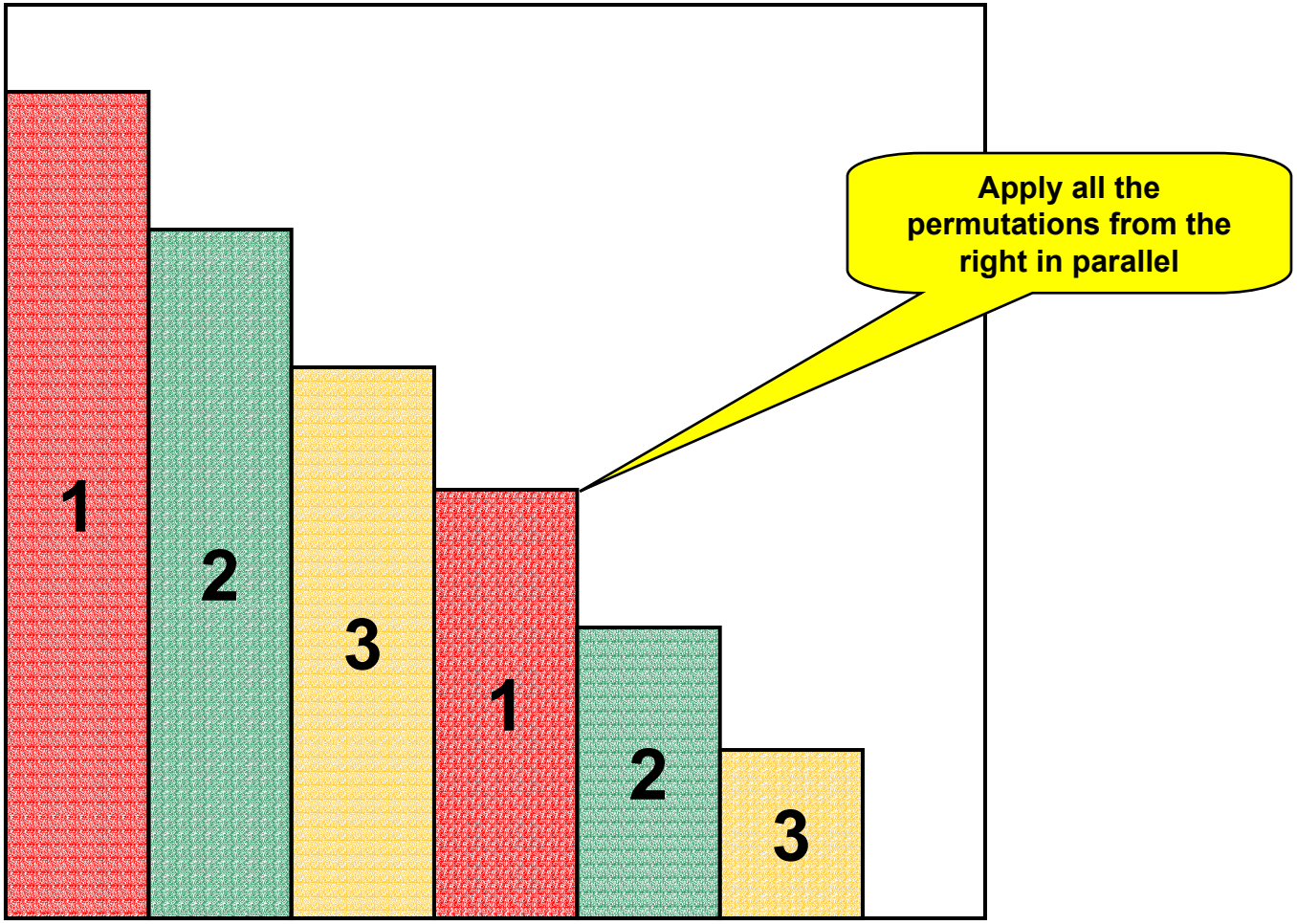
AMD DEVELOPER symposium 2002

- Factorizing the pivot block involves Level 2 BLAS (quasi-serial step)
- Permuting the rows (terrible for cache!)

LU Factorization: SMP Style



LU Factorisation: SMP Style (2)



Lessons

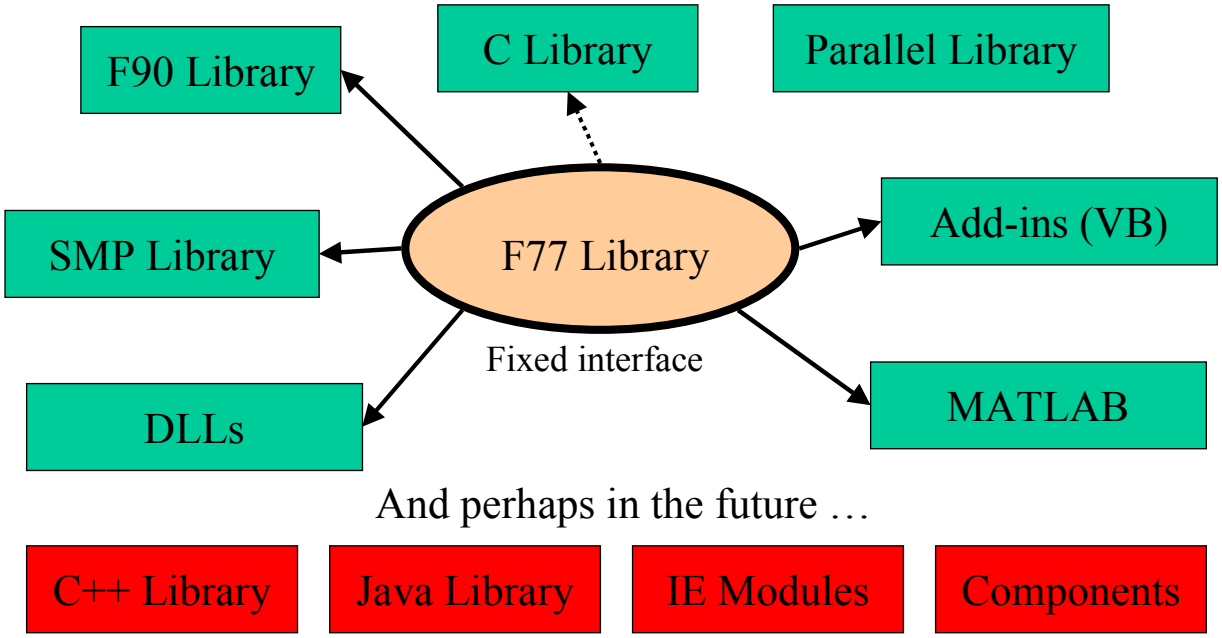
AMD DEVELOPER symposium 2002

- Pivot block now done at same time as other work (look ahead)
- Save up the permutations till the end – massively reduced cache-thrashing
  - E.g. 30% to under 0.5% computation time on PowerChallenge*!
- NAG has done it extremely efficiently and …
- … portably! Wide range of platforms
- On-going performance improvements
- Just re-link!

* Names and brands may be claimed as the property of others

Fin de siècle programmer

NAG Libraries - Previous situation

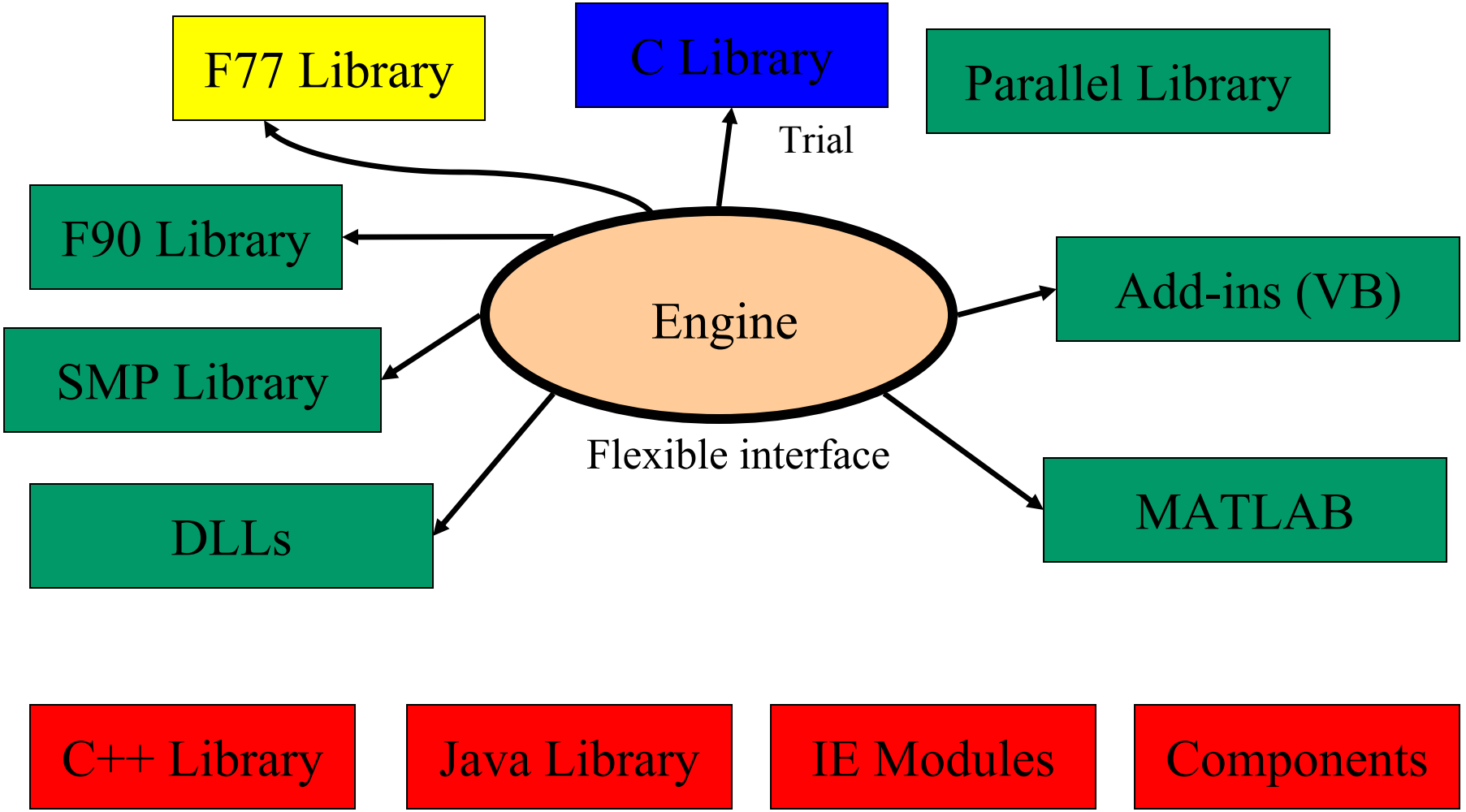


NAG Engine Approach

AMD DEVELOPER symposium 2002

- F77 Library  NAG Engine.
- Engine provides all algorithmic functionality (single source base).
- F77 Library just another wrapper to Engine.
- Free Engine interface and add hooks to simplify wrapping process.
- Automate wrapper generation from XML description.
- Add HPC versions directly into Engine.

Using the Engine



Engine Work

AMD DEVELOPER symposium 2002

- Code clean-up
- Thread-safety
- Intent statements.
- F77 Library wrappers
- Workspace size calculation
- Error message string
- Other I/O issues
- Hooks for callbacks
- Engine stringent test programs
- Engine examples

Essential ingredients

AMD DEVELOPER symposium 2002

- CVS
- XML / XSL
- GNUmake
- valgrind
- Automatic build/test procedure
- Engineers to fix problems!

XML routine documentation

AMD DEVELOPER symposium 2002

- XML processed to produce either Fortran or C documentation.
- Specs in documentation used by XSL to generate a C code wrapper to each Fortran engine routine
- Process is automatic

valgrind

AMD DEVELOPER symposium 2002

- Memory debugging tool developed by Julian Seward – available under GNU GPL
  http://developer.kde.org/~sewardj/
- Debugs ELF x86 executables under Linux
- Detects memory leaks, use of uninitialized data, etc.
- Related cachegrind program performs cache profiling

Advantages of Engine

AMD DEVELOPER symposium 2002

- Better quality code in F77 Library and other products.
- Quicker route to full coverage in new Libraries.
- Thread-safety in all Libraries.
- Better code maintenance and bug-tracking.
- More stringent tests.
- More time spent on algorithm development.
- Freedom to implement “behind-the-scenes” interface changes.

A disadvantage

AMD DEVELOPER symposium 2002

Anyone (including non-programmers) changing the XML documentation can affect C wrapper code, and hence library functionality.

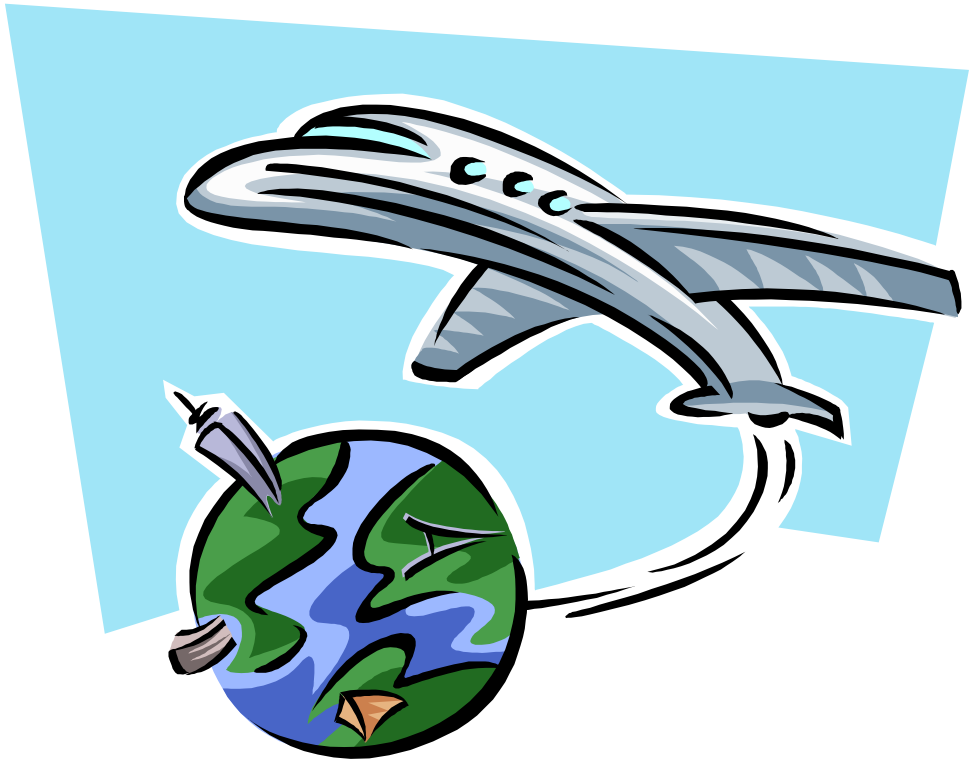
Therefore, we use an automatic build procedure.

Automatic build procedure

AMD DEVELOPER symposium 2002

- Checks out entire CVS repository
- Builds C wrappers from descriptions in XML doc
- Creates machine-dependent code (e.g. machine arithmetic description functions)
- Compiles all engine code and wrapper code
- Runs engine and wrapper test programs with valgrind
- Reports results to engineers by email

Procedure runs nightly in under 3 hours on a Linux machine powered by an AMD Athlon™ 1.2 GHz processor.

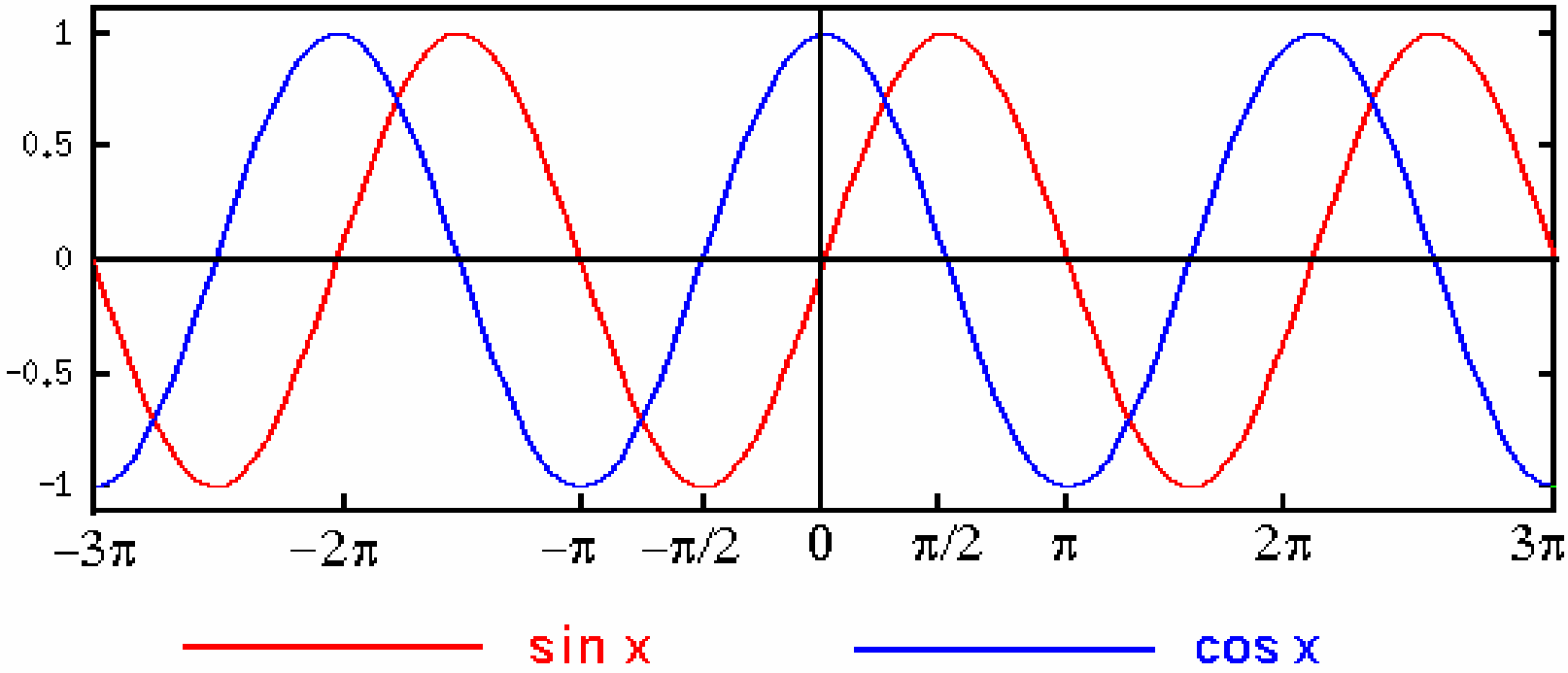


Elementary functions

AMD DEVELOPER symposium 2002

For elementary functions, like sin(x), cache is of much less concern. However, the speed of Hammer's double-precision floating-point and 64 bit integers gives us an advantage.

Periodicity of sin(x) and cos(x)



$\sin(x) = \sin(x + 2\pi) = \sin(x + 4\pi) = \ldots$

$\sin(x) = \cos(x - \pi/2)$

Argument reduction

Primary range is $[-\pi/4, \pi/4]$. If x does not lie in that range:

- Reduce argument x into r in primary range by subtracting correct multiple of $\pi/2$
- Depending on the multiple, evaluate $\sin(x)$ as $\sin(r)$ or $\cos(r)$ with sign adjusted
- For r in primary range, evaluate using a fast polynomial approximation

Argument reduction (cont)

Problem: π is not a representable number – finding the right multiple is *difficult,* especially when argument *x* is large

Solution (*Payne and Hanek, Radian reduction for trigonometric functions, SIGNUM Newsletter 18:19-24, 1983*): store $2/\pi$ in extra precision. For IEEE double precision arithmetic, about 500 bits are adequate

We can take advantage of Hammer's 64 bit integer arithmetic for efficiency in the extended precision multiplication

Speed of sin(x) on Hammer

AMD DEVELOPER symposium 2002

sin(*x*), cos(*x*):

Primary range: ~ 65 cycles
Largest arguments: ~ 300 cycles

All results accurate to less than 1 ulp

Trademark Attribution